# Introducción a la programación
# Introducción al entorno Haskell

Taller de Álgebra I

Segundo cuatrimestre de 2015

# Taller de Álgebra I

- ¿De qué se trata el taller de Álgebra I?
  1. Dar una introducción a la computación y a la programación, apta tanto para estudiantes de computación como para estudiantes de matemática.
  2. Programar los algoritmos que se ven en las teóricas y prácticas. . .
  3. . . . usando programación funcional.
     - ¿qué? ¿por qué?
- 3 horas semanales de clase teórico-práctica (modalidad taller)
- En los laboratorios de Computación.

# Régimen de cursada y evaluación

- Régimen de evaluación:
  - Un parcial a mediados del cuatrimestre que contendrá algún ejercicio de los vistos en clase y alguno nuevo con dificultad similar.
  - Un trabajo práctico en grupos de exactamente tres personas a fines del cuatrimestre.

  - En caso de no aprobar en alguna de las instancias:
    - Recuperatorio del parcial al final del cuatrimestre.
    - Recuperatorio del tp al final del cuatrimestre.

  - **Importante:** En las instancias de evaluación sólo podrán utilizarse las técnicas y herramientas vistas en clase.

- No es necesario aprobar el taller para anotarse en las materias correlativas.
- Sí es necesario aprobar el taller para rendir el final de Álgebra I.

# Listas de correo del taller

- Si te inscribiste por el SIU, deberías haber recibido el mail de bienvenida.
- Si no te llegan los mails o necesitás usar otra dirección, comunicate con nuestro Listmaster: `gino.scarpino (arroba) gmail.com`

- Anuncios para alumnos $\longrightarrow$ `algebra1-alu (arroba) dc.uba.ar`
  - Anuncios importantes sobre la cursada (¡leé tu mail seguido!)
  - Entre ustedes: para buscar grupo, armar grupos de estudio, etc.
- Consultas para docentes $\longrightarrow$ `algebra1-doc (arroba) dc.uba.ar`
  - Favor NO enviar consultas a `-alu`.
  - Cuando la respuesta es de interés general, nosotros cc'eamos a `-alu`.
  - Es mejor enviar tu consulta a `-doc` que a un docente en privado.

# ¿Qué hacemos en computación?

- Resolvemos problemas... ¿pero cómo?
- Para resolver un problema
  1. Identificar el problema a resolver.
  2. Pensar una solución con ciertas características (algoritmo) que sirva para resolverlo.
  3. Implementar un programa que respete ese algoritmo.

Los programas pueden ser vistos como descripciones ejecutables de la solución de un problema. Ejecutables por...

- Una **computadora** es una máquina que procesa información automáticamente de acuerdo con un programa almacenado.

  Claude Bellavoine, *¿Qué es una computadora?* Editorial El Ateneo, 1969.

# Paradigmas de lenguajes de programación

¿Cómo es un lenguaje de programación? ¿Cómo se le dice a una computadora lo que tiene que hacer?

**Paradigma:** Definición del modo en el que se especifica el cómputo (que luego es implementado a través de programas).

- Representa una "toma de posición" ante la pregunta: ¿cómo se le dice a la computadora lo que tiene que hacer?.
- Cada lenguaje de programación usa herramientas de uno o más paradigmas.

**Lenguajes representativos:**

- Paradigma de programación imperativa: C, Ada, Clu, ...
- Paradigma de programación en objetos: Smalltalk, Ruby, ...
- Paradigma de programación orientada a objetos: C++, Java, Python, ...
- Paradigma de programación en lógica: Prolog, Oz, ...
- Paradigma de programación funcional: LISP, Scheme, ML, F#, Haskell, Erlang, Clojure, Scala, ...

# Haskell: programación funcional

Haskell B. Curry (1900–1982)

- Existen muchos otros lenguajes de programación y otros paradigmas de lenguajes de programación, cada uno adaptado a diferentes situaciones.

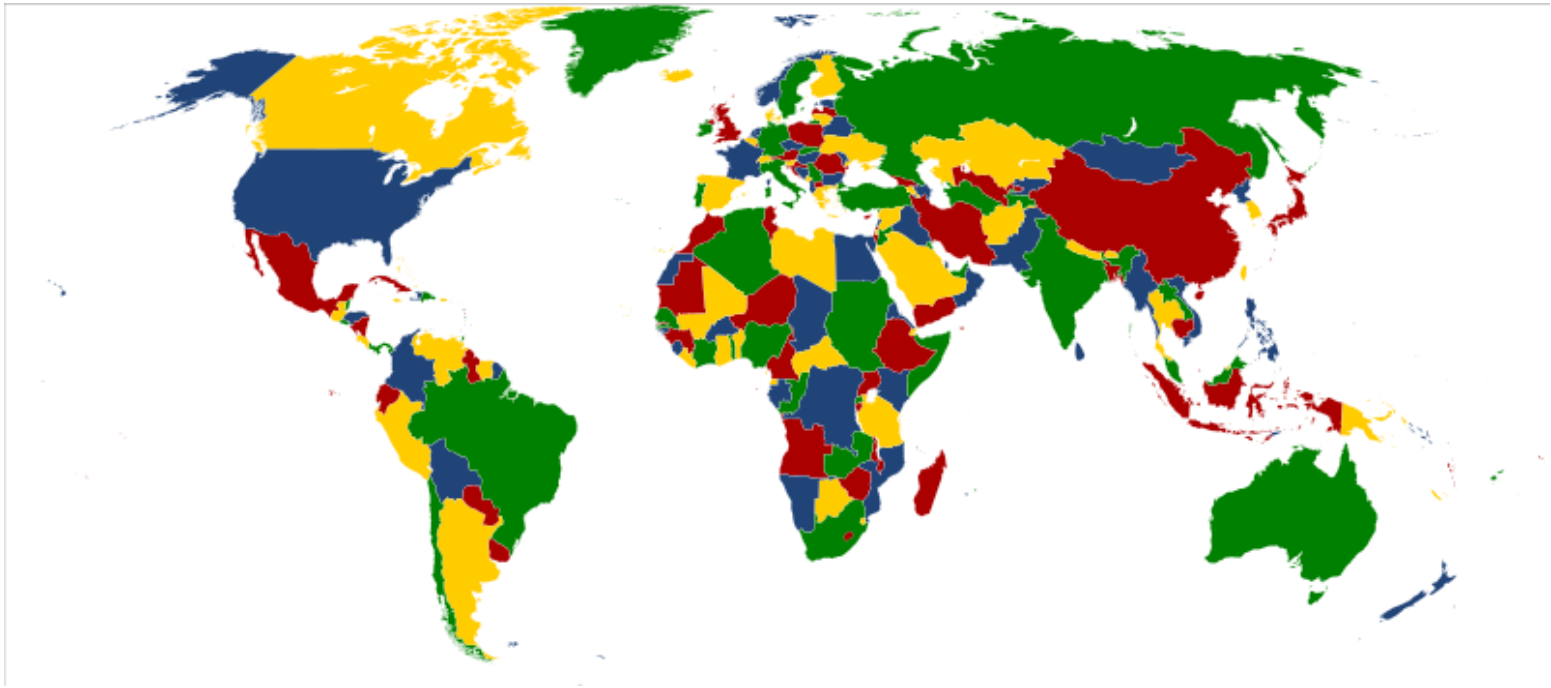
# ¿Por qué la computación es útil para un matemático?

La computación comenzó como una rama de la matemática, como una forma de entender (entre otras cosas) si todos los problemas son resolubles. Estas computadoras eran ideales. Con el tiempo, las computadoras se volvieron reales. Y cambiaron el panorama, la forma de hacer matemática. Permiten

- hacer cuentas que llevarían años de manera manual
- probar conjeturas
- chequar todos los casos en un teorema
- probar que un teorema está bien demostrado
- y más

# Veamos un ejemplo: los cuatro colores

¿Cuántos colores hacen falta para colorear un mapa de tal manera que dos regiones limítrofes tengan distintos colores? Ejemplo: el mapa del mundo y sus países se puede colorear con 4 colores.



En 1852, Francis Guthrie le preguntó a su hermano matemático si 4 colores alcanzarían para cualquier mapa. Ni el hermano de Guthrie, ni su profesor (Augustus de Morgan) ni otros matemáticos de la época pudieron demostrar que fuese así (ni encontrar un contraejemplo).

# Cuatro colores

En 1879, Alfred Kempe publicó un trabajo donde demostró que era cierto, que 4 colores bastaban para cualquier mapa.
En 1890, Percy Heawood probó que la demostración de Kempe estaba mal.
En 1977, Appel y Haken demostraron que era cierto, que 4 colores bastaban para cualquier mapa.
Pero la demostración de Appel y Haken dividió a la comunidad matemática. El problema:

- Se reducían a mano todos los mapas a unos 10.000. Chequear que estos eran todos los casos posibles no es sencillo.
- Además, en la verificación de estos casos, se encontraron varios errores (que fueron corregidos).
- Por último, Appel y Haken probaron que en estos casos 4 colores bastaban usando una computadora.

# ¿Por qué creerle a una computadora?

Un programa, decían los críticos de la computación entre los matemáticos, puede

- fallar (algo que se testea corriendo el mismo programa en distintas máquinas),
- y mucho más importante: estar mal hecho (tener *bugs*).

En 2004, Georges Gonthier demostró formalmente el teorema de los cuatro colores. Lo hizo con una computadora (no los 10.000 casos sino todo el teorema) pero escribiendo la demostración en un lenguaje que sirve para verificar demostraciones: Coq. **Observación**: Coq no es el único lenguaje desarrollado para verificar demostraciones. En el paper
`http://www.cs.ru.nl/~freek/comparison/comparison.pdf`
se muestran programas que prueban que $\sqrt{2}$ es irracional en 17 lenguajes distintos.

# Prueba automática de teoremas

- Es posible escribir un algoritmo que permite probar mecánicamente teoremas?
- Más precisamente, dar un algoritmo general que dada una fórmula de la lógica de primer orden indique si la misma es un teorema o no.
- *Entscheidungsproblem* (problema de decisión) formulado por David Hilbert en 1928.

## Problema de decisión

Dados:

- un conjunto $S$ de datos de algún tipo (por ejemplo, fórmulas de la lógica de primer orden), y
- una propiedad $P$ de los elementos de $S$

El problema de decisión asociado es

- dar un algoritmo que termina y devuelve verdadero o falso para cada elemento $s \in S$, y retorna verdadero si y sólo si $s$ tiene la propiedad $P$.

Un problema es decidible si existe tal algoritmo

# Decidibilidad del Entscheidungsproblem

- Se intentó encontrar la solución al Entscheidungsproblem. . . sin éxito.
- Luego la pregunta fue: ¿y si se puede probar que tal algoritmo no existe?
- Independientemente Turing y Church en 1936/7 probaron que no puede existir. El problema es *indecidible* (no tiene solución).
  - Dieron una definición precisa de algoritmo.
  - Turing desarrolló las Máquinas de Turing y probó que el problema de la parada (*halting problem*) no es decidible.
  - Church desarrolló el cáculo lambda y mostró que no existe algoritmo capaz de decidir si dos funciones son equivalentes.
  - Estos enfoques son equivalentes y capturan la noción de los problemas que son computables.

# Recursos para el Taller

## Interprete de Haskell

GHCI (The Glasgow Haskell Compiler Interactive environment)

- Ubuntu? sudo apt-get install ghc
- Mac? brew install haskell-platform
- Windows? (buu) http://www.haskell.org/ghc/download

## Editores de Texto

- Ubuntu? Sublime, Gedit, Geany, vim, etc.
- Mac? Sublime, Textmate, Atom, TextEdit, vim etc.
- Windows? Sublime, Atom, Notepad++, etc.

Trabajen **cómodos**!!

# Más Recursos

## Cómo compartir Código

1. ~~Email~~ (no)
2. Dropbox (puede ser)
3. Google Docs (mejor)
4. svn (se hizo la luz)
5. git (vamoo los pibe)

## Comandos útiles para consolas

**Qué es una consola???**

- cd
- ls (dir en windows)
- cp, mv, rm (copy, move, del en windows)
- pwd

# Ejercicio del día

1. Crear un archivo de texto con el siguiente contenido:
   `suma x y = x - y`
   (sí, sabemos que está mal)
2. Guardarlo con el nombre **clase0.hs** en alguna carpeta (no se olviden la ubicación)
3. Abrir GHCI (ghci desde la consola o ejecutando el programa en windows)
4. Cargar el archivo: `:l <ruta del archivo>` (aquí pueden arrastrar el archivo hasta la consola para obtener la ruta)
5. Dentro de GHCI, ejecutar lo siguiente: `suma 2 3`
6. Corregir y guardar la función en el archivo para que efectivamente sea la suma (sin cerrar la consola de GHCI)
7. Recargar el programa: `:r`
8. Ejecutar `suma 2 3` y ver que de 5 :)
9. Si quieren, pueden cerrar el intérprete ejecutando: `:q`